Secure Computer Systems
Secure Distributed Data Repository

Manish Choudhary ***Sakshi Sharma***
Georgia Institute of Technology
Atlanta, Georgia
{mchoudhary8, ssharma311}@gatech.edu

Introduction

The secure distributed data repository is a client-server system which facilitates the users to store their files on a central repository ensuring optional confidentiality and integrity of the documents. The service provides a way to establish secure and mutually authenticated communication channel which is used to exchange the request and response messages. The users are provided with different options such as retrieving/storing the files from/on the server and delegating access rights to other users.

Key Protocols and Functionality Implementation

1. Certificate Management

Openssl has been used to create a certification authority by generating its certificate and private key. Certificates have been signed for all the members of this system using the private key of the CA. It is assumed that all the entities have the certificate of the CA which can be used to verify the trustworthiness of the certificate received from any entity in the system.

2. Session Establishment Service

The protocol implemented to establish a secure and mutually authenticated communication channel is similar to the Diffie-Hellman key exchange protocol.
The server listens for the connect requests on the service port. When the client requests for a connection following steps are performed:

- Server sends its certificate to the client requesting the secure session
- Client verifies that the server certificate has been signed by the trusted CA
- Client sends its certificate to the server
- Server verifies that the client certificate has been signed by the trusted CA
- Server generates a random 128 bit key and sends it to the client after encrypted with the client's public key
- Client retrieves the key sent by the server by decrypting the message with its private key

- Client generates a random 128 bit key and sends it to the server after encrypted with the server's public key
- Server retrieves the key sent by the client by decrypting the message with its private key
- Both the server and the client generate the session key by computing the hash of the two random keys exchanged.

This completes the mutual authentication and key exchange protocol. All the requests and responses are exchanged between the client and the server over a secure changed encrypted using the session key generated.

3. GET Service

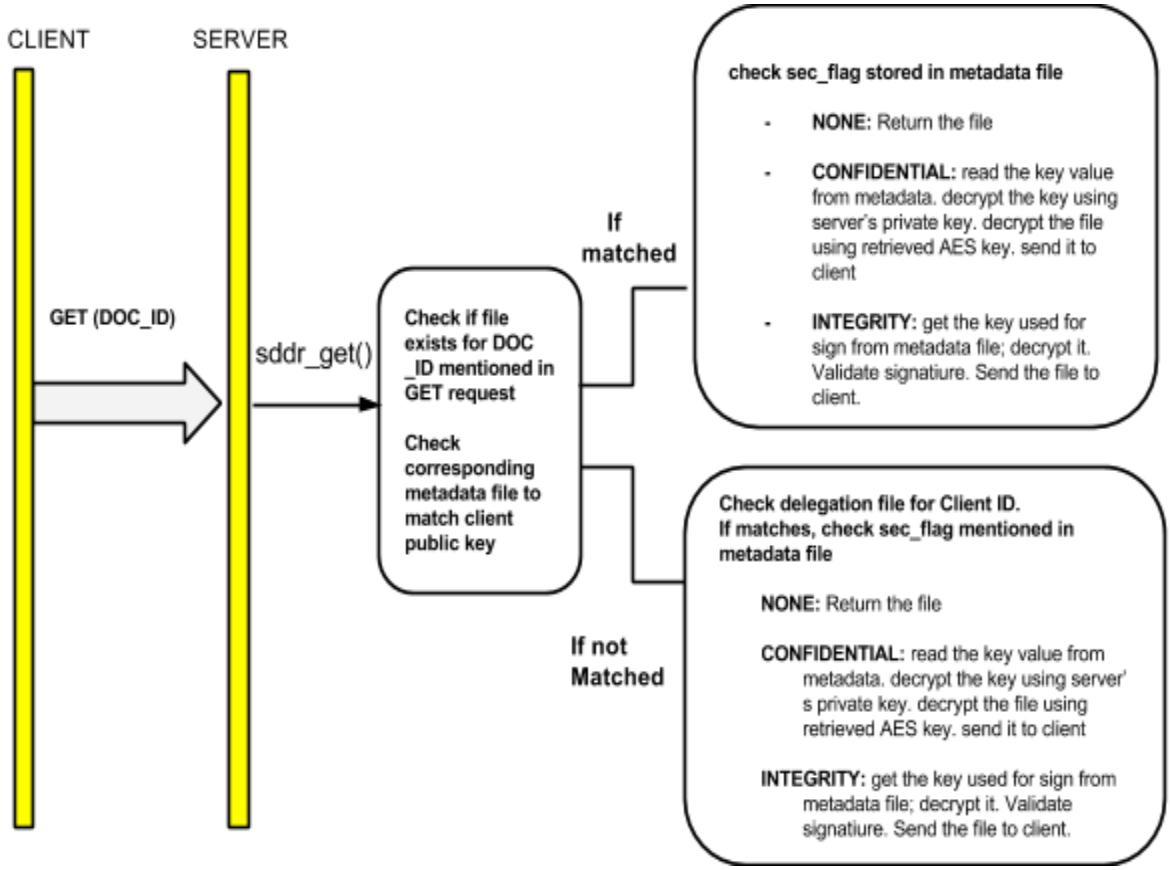
The user can request to fetch a file stored at the central repository by providing the UID of the file which is assumed to be known to the client. The GET request has the following format.

@GET:<UID>

This request gets encrypted and is sent to the server. The server extracts the parameters from the formatted request after decrypting it using the session key. Then, it checks its local database for the file with uid as UID. If the document is present, the meta data file for that document is looked up to match client public key. If it matches, based on the SEC_FLAG associated with the file, the contents of the file are returned to the client. Otherwise, the metadata file containing the delegation rules is checked. Following checks are made to ensure that the client making the request has the delegated authorization to retrieve the file:

- The stored client ID in some entry is either equal to "ALL" or is equal to the ID of the client making the request
- The delegated right present in the entry is either equal to "BOTH" or is equal to "GET"
- The validity period mentioned by the delegator has not expired.

If the client is authorized, the data is returned to the client over an encrypted channel and a file gets created in the client's local repository with this data.



4. PUT Service

The user can request to store a file at the central repository by providing the filename and the security property (None, Confidentiality or Integrity) required. If the file was never stored on the server before, it can be given any name. Otherwise, it should be stored with the name <UID>.txt everywhere. The request has following format.

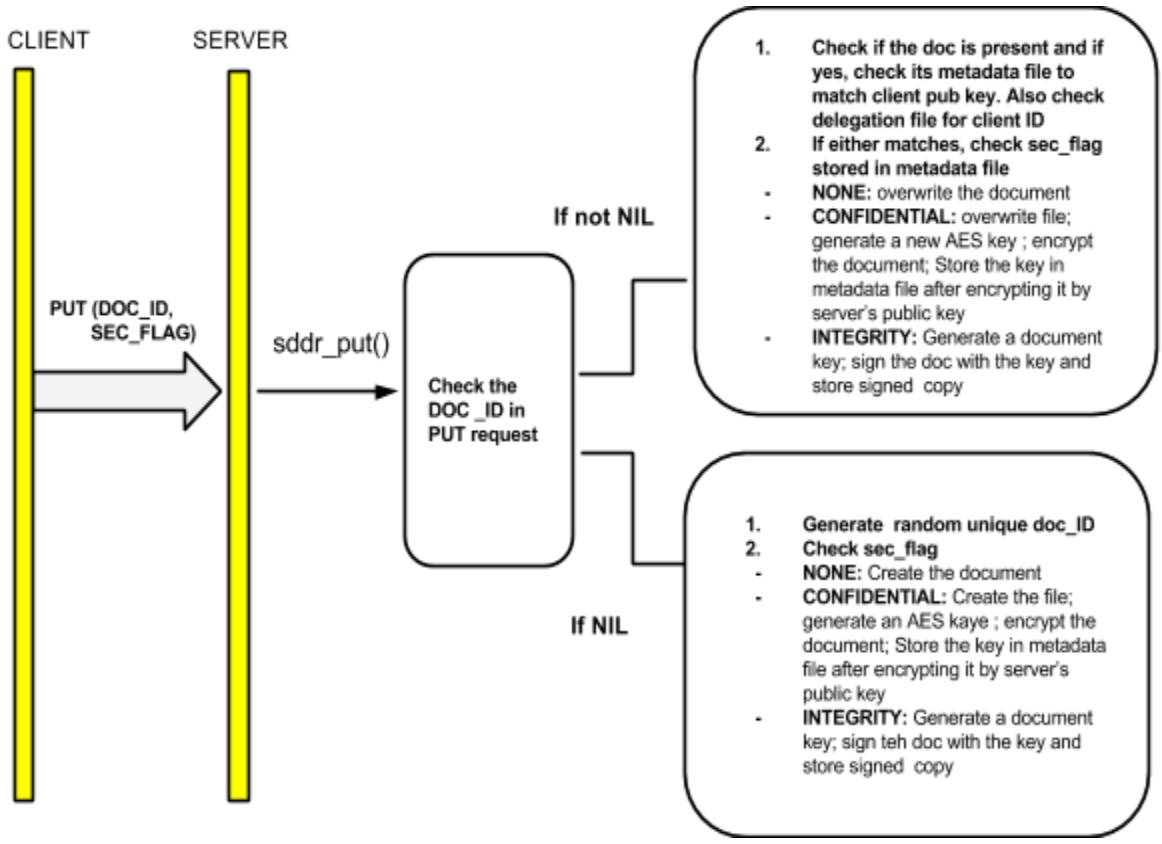
@PUT:<Security Flag>:<UID>:<file data>

This request is sent to the server over an encrypted channel which extracts the parameters from the decrypted form of the received request. If the UID in the request is “NIL”, it generates a new random UID for this file. The requestor is assumed to be the owner of the file which gets stored on the server after processing its content based on the security flag. The metadata file is created with following format.

<UID> <CLIENT_PUB_KEY> <SECURITY_FLAG> <KEY>

The parameter <KEY> denotes the value of AES key encrypted using server’s public key if the Sec_Flag is CONFIDENTIAL else it is marked as “NONE”. Hash is also added in case of INTEGRITY.

If the UID is specified, the presence of file is checked and it is ensured that the requestor is either the owner or has the delegated rights to update the file within the validity period. If allowed, the file gets stored on the server after processing the data based on the security flag and the corresponding metadata file is also replaced ensuring that the owner gets updated only if the PUT request from the owner.



5. Delegation Service

This service allows the user to delegate some or all of the access rights for a particular file to an individual or all the other clients in the system for a certain validity period. The user is asked to provide the file UID, the client ID ("ALL" if wants to delegate to all), the rights to be delegated(GET/PUT/BOTH), the validity period and the propagation right(allowed or not). The delegation request has the following format:

@DEL:<UID>:<delegateToClient>:<GET/PUT/BOTH>:<Validity Period>: <Propagation Flag>

This request is sent to the server over an encrypted channel which extracts the parameters from the decrypted version of the request received. Then, it is checked that the requesting client is the owner of the file or has the delegated rights for the same file and the access rights that he wants the further delegate with propagation flag set to 1. If authorized, a new delegation rule with following format gets added to the delegation related metadata file.

<clientID/ALL> <GET/PUT/BOTH> <Delegation request time> <Validity Period> <Propagation Flag>

6. Session Termination Service

This service allows the user to terminate the current session. The user is asked to confirm the termination. If confirmed, all the files retrieved by the client from the server during this secure session gets updated to the server along with the corresponding metadata files. A request of following format is sent to the server to inform that the client wants to terminate the session.

@TER:

On receiving the request, the server terminates the current session after destroying the session key and closing the socket. Finally, the client terminates after destroying the session key and closing the socket.

Note: The purpose of each individual function has been explained in the code itself and thus, is not present in this report to avoid redundancy. Please refer the different modules to understand the purpose of each module and the functions helping to achieve that purpose.

Individual Contribution

The individual contribution of each team member has been mentioned below.

Design
Both have worked together to design the system.

Implementation
The server side GET and PUT related functionality present in calls.c module has been implemented by Sakshi. All other functionality in the system has been implemented by Manish.

Integration and Testing
Both have worked on integrating the modules.

Security Evaluation
Both have worked together to perform the security evaluation of the code to identify the potential exploitable vulnerabilities.

Security Evaluation

The code has been analyzed using Flaw Finder tool to find any exploitable vulnerabilities in the code. No critical vulnerabilities were found during the evaluation. Safe functions implementing the length check have been used to avoid buffer overflow scenarios. Bound checking has been performed on all the inputs taken and thus, avoids the vulnerabilities which can be be introduced by the user input. So, there are no critical vulnerabilities in the code which are exploitable in this context.